Reflection

Architecture of server was restructured after the planning phase. The biggest change was in separating logic and data classes.

Managers
Originally, I wanted to store logic or manager class together with its data classes, but after getting the feedback I realized that it would be better to store logic and data related classes in two different packages.

Business Rules
Based on Open Closed Principle and to reach better level of abstraction, I use two Interfaces, one for map related rules and one for player related. These interfaces are later extended by the real specific business rule classes.

Also, I used Test Driven Development principle during implementation of Map related business rule classes. It helped to implement the classes with better understanding what to expect and debugging phase was short, even though the implementation phase took a bit longer.

Exceptions
I created additional Exceptions for specific cases. All exceptions extend GenericException so Spring boot processes them respectively. All the exceptions are stored in the same package.

Abstract classes
During planning phase to avoid code dublication and adhere to DRY and OCP principles, I introduced generic abstract classes GameMap and GameMapTile. To additionally increase flexibility of code. These classes were supposed to be extended by FullMap and HalfMap classes. But during implementation I realized that HalfMap class was not necessary and was makin code more complex, so I was using only FullMap related classes. I still call GameMap class to create better abstraction and for possible future extentions, but for now GameMap class is extended only by FullMap class.

Data sharing
In the context of sharing data between the Manager classes, I decided on store shared data in the GameEntity class, in the face of avoiding data duplication, neglecting using global variables, to achieve better reusability. GameEntity class is the "only source of truth" for logic classes. GameEntity class also helps to track and control all the games.